Viability-Based Guaranteed Safe Robot Navigation

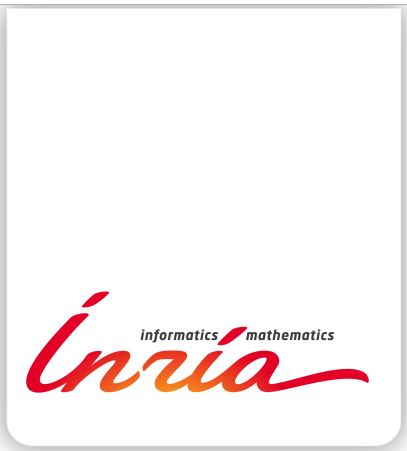
Mohamed Bouguerra, Thierry Fraichard, Mohamed Fezari

Viability-Based Guaranteed Safe Robot Navigation

Mohamed Bouguerra, Thierry Fraichard, Mohamed Fezari

RESEARCH REPORT
N° 9217
October 2018
Project-Team PERVASIVE

ISSN 0249-6399 ISRN INRIA/RR--9217--FR+ENG



Viability-Based Guaranteed Safe Robot Navigation

Mohamed Bouguerra*, Thierry Fraichard†, Mohamed Fezari*

Project-Team PERVASIVE

Research Report n° 9217 — October 2018 — 22 pages

Abstract: Guaranteeing safe, *i.e.* collision-free, motion for robotic systems is usually tackled in the Inevitable Collision State framework. This paper explores the use of the more general Viability theory as an alternative when safe motion involves multiple motion constraints and not just collision avoidance. Central to Viability is the so-called *viability kernel*, *i.e.* the set of states of the robotic system for which there is at least one trajectory that satisfies the motion constraints forever. The paper presents an algorithm that computes off-line an approximation of the viability kernel that is both conservative and able to handle time-varying constraints such as moving obstacles. Then it demonstrates, for different robotic scenarios involving multiple motion constraints (collision avoidance, visibility, velocity), how to use the viability kernel computed off-line within an on-line reactive navigation scheme that can drive the robotic system without ever violating the motion constraints at hand.

Key-words: Mobile robotics; Autonomous Navigation; Collision Avoidance; Viability Theory; Guaranteed Safety;

* University of Annaba, Algeria
† Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble, France

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Navigation pour robot avec garantie de sécurité basée sur la théorie de la viabilité

Résumé : La garantie de mouvement sans collision pour les systèmes robotiques est généralement abordée dans le cadre des Etats de Collision Inévitable. Cet article explore l'utilisation de la théorie plus générale de la Viabilité comme alternative lorsque le mouvement implique des contraintes de mouvement autres que l'évitement de collision. Le *noyau de viabilité*, *i.e.* l'ensemble des états du système robotique pour lequel il existe au moins une trajectoire qui satisfait à jamais les contraintes de mouvement, est un élément central de la théorie de la viabilité. Cet article présente un algorithme qui calcule hors ligne une approximation du noyau de viabilité qui est à la fois conservative et capable de gérer des contraintes dynamiques telles que des obstacles mobiles. Ensuite, il démontre, pour différents scénarios robotiques impliquant plusieurs contraintes de mouvement (évitement de collision, visibilité, vitesse), comment utiliser le noyau de viabilité calculé hors ligne dans un schéma de navigation réactive en ligne capable de piloter le système robotique sans jamais violer les différentes contraintes de mouvement.

Mots-clés : Robotique mobile; Navigation autonome; Evitement de collision; Théorie de la viabilité; Sécurité garantie;

Contents

1 Introduction

It is expected that robotic systems of various kinds and with various roles will increasingly live in human-populated workspaces, the *safety* of their motion grows in importance accordingly. Safe motion is often equated with the ability of the robot to avoid collision with its surroundings. It is now understood that collision avoidance requires the ability to stay away from what is known as *Inevitable Collision States* [12]. An Inevitable Collision State (ICS) is a state for which no matter what the future trajectory of the robot is, collision will eventually occur. Designing a control system for a robot that is able to compute its inevitable collision states and stay away from them at all time is the key to *guaranteed motion safety* [23].

The characterization of ICS is intricate since it requires in theory to check for collision every possible trajectory of infinite duration a robot might follow from a given state. A practical answer to this issue is to select a subset of so-called *evasive trajectories*, and a state is deemed an ICS if none of the evasive trajectories starting from it is collision-free. This results in a conservative approximation of the ICS set whose quality depends essentially on the choice of the evasive trajectories. If not for a good choice, most

states may end up labeled as ICS. The problem with this approach is that it is not clear in general how to select the set of evasive trajectories so as to obtain a reasonable approximation of the ICS set. In static workspaces, braking trajectories, *i.e.* trajectories that drive the robot to a stop, are good candidates. In dynamic workspaces though, even with knowledge of the future behavior of the obstacles, it is all the more challenging to determine an appropriate set of evasive trajectories.

Now, there is often more to motion safety than mere collision avoidance. In many cases, the motion of the robot must satisfy various types of constraints in order to be considered safe. For instance, a legged robot should maintain its balance, the speed of an airplane should never go beyond its stalling speed, or a spy robot should stay out of sight of a patrol. No matter what the set of constraints that the robot ought to satisfy, they yield a set of forbidden states that the robot should avoid. The tenet of this paper is that motion safety in general should receive an ICS-like treatment. In other words, the robot should of course avoid the states that violate the constraints, but more importantly, it should avoid the states inevitably leading to them.

To address this key question, this paper considers the *Viability* framework which is more general than the ICS framework. Viability theory [1] addresses the following question: how to control dynamical systems subject to *viability constraints*? Viability constraints define a subset of the state space of the system within which the system should remain. A *viable* state is guaranteed to have at least one sequence of controls which will keep the system within the viability constraint set indefinitely. Conversely, *nonviable* states are those where failure is no longer avoidable. Note that when collision avoidance is the only viability constraint, the nonviable states are Inevitable Collision States. The *viability kernel* of the viability constraints is the set of all its viable states. In this framework, the ability to design a control system for a robot that is able to compute its viability kernel and remain inside it at all times is also the key to guaranteed motion safety.

The characterization of the viability kernel is at least as challenging as the characterization of ICS and the approach adopted in this paper is also to compute a conservative approximation of the viability kernel. The starting point of this paper is an algorithm originally proposed to approximate viability kernels [32]. This paper builds upon and expands [5], the primary contribution of this paper is the **Conservative Viability Algorithm**, an adaptation of [32]'s algorithm designed to make it, (i) *conservative*, and (ii) able to handle *time-varying viability constraints* such as moving obstacles. Besides its ability to handle different kinds of safety constraints in a unified manner, the key advantage of the algorithm proposed herein is that, unlike its ICS counterpart, the quality of the conservative approximation is in no way dependent on the choice of an appropriate set of evasive trajectories. This feature comes at the cost of a greater computational complexity though. To demonstrate its versatility and its ability to address different kinds of viability problems, the algorithm proposed has been implemented and used to compute the viability kernel for the case of a double integrator robot in seven scenarios with mixed combinations of workspace (static, moving obstacles) and viability constraint types (collision avoidance, visibility, velocity). To demonstrate the usefulness of the viability kernel computed off-line by the Conservative Viability Algorithm, it is used on-line inside a basic and purely reactive navigation scheme that proved able to control the robot in the different scenarios without *ever* violating the viability constraints at hand. Note that the viability kernel could just as well be used inside a motion planner in order to compute safe motions.

The paper is organized as follows: §2 reviews the relevant literature. Viability theory and the viability algorithm of [32] are respectively presented in §3 and §4. Then, §5 and §6 respectively describe how to transform the original algorithm into a conservative one able to handle time-varying viability constraints. Finally, the robotic scenarios demonstrating guaranteed safe navigation in different situations are presented in §7 and §8

2 Related Works

On the ICS front, although the details may vary, most of the proposed ICS approximation methods rely in essence on the same principle: a subset of evasive trajectories is selected and states are labeled ICS if none of the evasive trajectories are collision-free. In static workspaces, braking trajectories (which drive the robot to a stop), are an obvious choice [2, 35, 33]. When the robot cannot stop, *e.g.* it is an airplane, circling trajectories have been considered [34, 3]. In dynamic workspaces, imitating trajectories (which maintain zero relative velocity with the obstacle), have been proposed in [23]. Whatever the subset of evasive trajectories selected, it is difficult to ensure the quality of the approximation for all situations and not end up with most states conservatively labeled as ICS. This difficulty plus the fact that absolute motion safety requires in general to reason over an infinite time horizon [13] have led some authors to settle for weaker motion safety guarantees. For instance, [6] introduces *passive safety*: it guarantees that if a collision occurs, the robot will be at rest. A stronger form, *friendly passive safety* [21, 26] ensures that if a collision ever happens the robot will be at rest, and the obstacles could have avoided the collision if they wanted to. Other methods settle to even less, they aim to improve the chance of surviving collisions with no strict guarantees however. They use other types of trajectories: for instance, trajectories that are guaranteed to be collision-free only up to a finite time [14, 15], or trajectories that are collision-free with respect to one obstacle at a time, instead of considering them all at once [8, 37].

On the viability front, several methods have been proposed for the approximation of the viability kernel. For low dimensional systems with non linear dynamics, there are discrete methods such as the viability algorithm [32], those based on the viscosity solutions for Hamilton-Jacobi partial differential equations [25, 20], or most recently interval analysis [27]. For systems whose dynamics can be described as polynomials, more efficient methods based on invariance sets have been used [36, 18]. For linear systems with higher dimensions, Lagrangian techniques can be exploited as in [22]. Viability Theory has seen applications in several fields, and mobile robotics is one of them. In [4], the problem of a biped that has to maintain its balance while also ensuring passive safety has been addressed using Model Predictive Control. Closer to the work proposed herein, a discrete method based on [32] was developed in [19] for the purpose of safe autonomous racing, *i.e.* to drive as fast as possible around a predefined track. In [16] and [17], machine learning has been deployed to approximate the viability kernel for mobile robots. The purpose in [17] was to filter out unsafe states from the search space, to speed up motion planners, while in [16], it was to help augmenting systems' safety by preventing them from entering failure regions. A learning approach is prone to misclassification, which may not be a problem in the first case, but would void safety guarantees in the latter one.

3 Viability Theory

This section briefly recalls the key concepts of the viability theory, the reader is referred to [1] for more details. Viability theory is a set of mathematical techniques that address this specific question: how to control dynamical systems subject to *viability constraints*? Viability constraints generally define a subset of the state space of the system within which the system should remain, *e.g.* avoiding collisions for a mobile robot, remaining dynamically balanced for a legged robot. A *viable* state is guaranteed to have at least one sequence of controls which, when applied from said state, will keep the system from failure, *i.e.* keep it within the viability constraint set indefinitely (Fig. 1). Conversely, *nonviable* states are those where failure is no longer avoidable. Note that when collision avoidance is the viability constraint then nonviable states are Inevitable Collision States [12]. The *viability kernel* of the constraint set is the set of all its viable states. These concepts can be formalized as follows.

Let $\mathcal{A}$ denote a continuous-time dynamical system whose dynamics is described by differential equa-

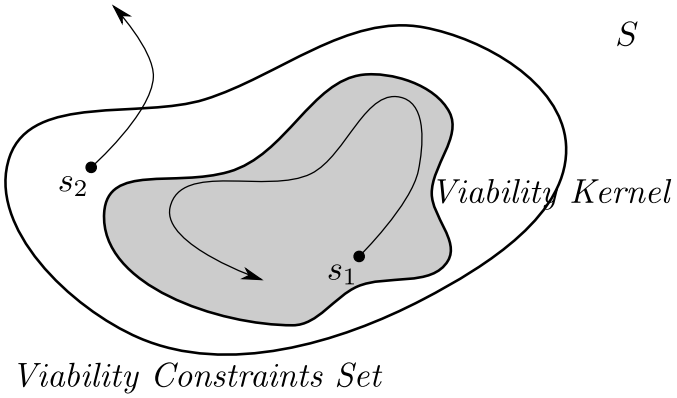


Figure 1: Main viability concepts: s_1 is viable, s_2 is nonviable.

tions of the form:

$$\dot{s}(t) = f(s(t), u(t)) \tag{1}$$

where $s(t) \in S$ is the state of $\mathcal{A}$ at time t. The state of $\mathcal{A}$ is influenced by a control $u(t) \in \mathcal{U}$ that can be state-dependent. S and $\mathcal{U}$ respectively denote the state space and the control space of $\mathcal{A}$. Viability constraints are characterized by the compact subset $\mathcal{K} \in S$ within which the system must be kept.

Let $\tilde{u} : [0, t_f] \longrightarrow \mathcal{U}$ denote a *control trajectory*, *i.e* a time-sequence of controls, t_f is the duration of $\tilde{u}$. The set of all possible control trajectories is denoted $\tilde{\mathcal{U}}$. Starting from an initial state $s(t_0)$ at time t_0, a *state trajectory* $\tilde{s}(s(t_0), \tilde{u})$ is derived from a control trajectory $\tilde{u}$ by integrating (1). $\tilde{s}(s(t_0), \tilde{u}, t)$ denotes the state reached at time t. A state trajectory $\tilde{s}(s(t_0), \tilde{u})$ is said to be viable in $\mathcal{K}$ on an interval $[0, t_f], t_f \leq +\infty$, if $\forall t \in [0, t_f]$, $\tilde{s}(s(t_0), \tilde{u}, t) \in \mathcal{K}$. *Viable* states are those for which there exists at least one control trajectory $\tilde{u}$ yielding a state trajectory viable in $\mathcal{K}$ at all times *i.e.* on the interval $[0, +\infty)$.

The basic problem in the viability theory is to find the *viability kernel* of $\mathcal{K}$, *i.e.* the set of all its viable states:

Definition 1 (Viability Kernel)

$$\mathit{Viab}_f(\mathcal{K}) = \{s(t_0) \in \mathcal{K} \mid \exists\, \tilde{u} \in \tilde{\mathcal{U}} : \forall t \geq 0,\ \tilde{s}(s(t_0), \tilde{u}, t) \in \mathcal{K}\} \tag{2}$$

Once the viability kernel is determined, the next step is to compute the *regulation map*, the set-valued map $s \in \text{Viab}_f(\mathcal{K}) \rightsquigarrow R(s) \subset \mathcal{U}$ that indicates at each state the set of *viable* controls that, when applied, will maintain the system inside the viability kernel.

The following section will present the *viability algorithm* from [32], that aims at approximating the viability kernel of $\mathcal{K}$ for the dynamical system (1) along with providing its corresponding regulation map.

4 Viability Algorithm

The viability algorithm from [32] operates in two stages. It first approximates the original continuous problem by discretizing it in time and space. Then, it computes the exact viability kernel for the discretized problem in an recursive way.

4.1 Discretization

The problem is first discretized in time. There exist different ways to transform a continuous-time model into its discrete counterpart, the Euler explicit discrete scheme is the most straightforward. Under this

scheme, the discrete-time version of the dynamical system (1) is:

$$\begin{cases} s_{n+1} = g(s_n, u_n) = s_n + \rho f(s_n, u_n) \\ u_n \in \mathcal{U} \end{cases} \tag{3}$$

where ρ is the discrete time step. The state space is then reduced to a finite subset of S, for instance a regular grid of step d, denoted S_d. Note that the discrete system (3) cannot be readily defined on the finite grid S_d because nothing guarantees that, for all $s \in S_d$, the image $g(s, u)$ belongs to S_d. To address this issue, g^r is introduced, it is the extension of g with an hyperball of radius r:

$$g^r = g + \mathcal{V}(r) \tag{4}$$

where $\mathcal{V}(r)$ is the hyperball of radius r. r is chosen such that:

$$\forall s \in S_d \, , \, g^r(s, u) \cap S_d \neq \emptyset \tag{5}$$

An obvious choice is $r = d$. The control space is also reduced to a finite subset denoted $\mathcal{U}_d$. Finally, the discrete and finite dynamical system obtained is:

$$\begin{cases} s_{n+1} \in g^r(s_n, u_n) = s_n + \rho f(s_n, u_n) + \mathcal{V}(d) \\ u_n \in \mathcal{U}_d \end{cases} \tag{6}$$

4.2 Computing the Discrete and Finite Viability Kernel

The viability kernel of $\mathcal{K}_d = \mathcal{K} \cap S_d$ for the discrete and finite system (6) is computed as follows: $\mathcal{K}^0$ is initialized to $\mathcal{K}_d$, and the sequence of subsets $\mathcal{K}^1, \mathcal{K}^2, \mathcal{K}^3, ..., \mathcal{K}^n, ...$ is recursively defined such that:

$$\mathcal{K}^{n+1} = \{s \in \mathcal{K}^n \mid \exists u \in \mathcal{U}_d \, : \, g^r(s, u) \cap \mathcal{K}^n \neq \emptyset\} \tag{7}$$

This will incrementally refine the grid $\mathcal{K}_d$ by discarding at each iteration the states from which the system will inevitably leave the grid in the next step. Let $\mathcal{K}^\infty = \bigcap_{n=0}^{\infty} \mathcal{K}^n$, it has been established in [32] that $\mathcal{K}^\infty$ is the largest subset of $\mathcal{K}_d$ such that:

$$\{\forall s \in \mathcal{K}^\infty \, , \, \exists u \in \mathcal{U}_d \, : \, g^r(s, u) \in \mathcal{K}^\infty\} \tag{8}$$

or equivalently:

$$\mathcal{K}^\infty = \text{Viab}_{g^r}(\mathcal{K}_d) \tag{9}$$

and, since $\mathcal{K}_d$ is finite, there exists a finite integer p such that:

$$\forall n \geq p \, : \, \mathcal{K}^n = \mathcal{K}^p \tag{10}$$

which guarantees the convergence of the recursion. Thus the viability kernel of $\mathcal{K}_d$ for the discrete and finite system (6) can easily be computed in a finite number of steps using (7) (see Algorithm 1).

Once the viability kernel $\text{Viab}_{g^r}(\mathcal{K}_d)$ has been computed, it is straightforward to retrieve the discrete regulation map R_d which is defined for every state in $\text{Viab}_{g^r}(\mathcal{K}_d)$ as:

$$R_d(s) = \{u \in \mathcal{U}_d \mid g^r(s, u) \in \text{Viab}_{g^r}(\mathcal{K}_d)\} \tag{11}$$

This regulation map provides all the viable controls that are available at each state. Choosing controls belonging to R_d ensures that the system will remain in $\mathcal{K}_d$ at all times. It is important to note that, although the viability kernel $\text{Viab}_{g^r}(\mathcal{K}_d)$ is only an approximation of the viability kernel for the continuous

Algorithm 1: Viability Algorithm [32]

Input: Discrete-time dynamical system g^r; Discrete state space S_d; Discrete control space $\mathcal{U}_d$; Viability constraint set $\mathcal{K}$

Output: Viability kernel $\text{Viab}_{g^r}(\mathcal{K}_d)$

$\mathcal{K}_d \leftarrow S_d \cap \mathcal{K}$;
$n \leftarrow 0$;
$\mathcal{K}^0 \leftarrow \mathcal{K}_d$;
repeat
  $\mathcal{K}^{n+1} \leftarrow \{s \in \mathcal{K}^n \mid \exists u \in \mathcal{U}_d(s) : g^r(s,u) \in \mathcal{K}^n\}$
  $n \leftarrow n+1$
until $\mathcal{K}^n = \mathcal{K}^{n+1}$;
return $\mathcal{K}^n$

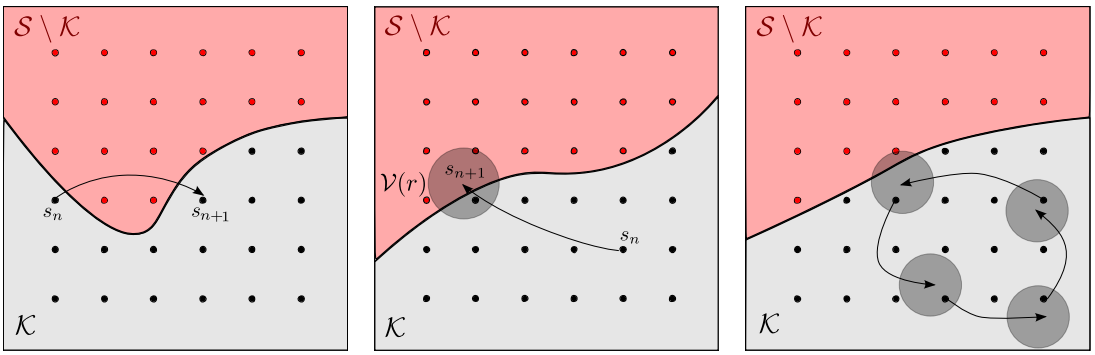
problem $\text{Viab}_f(\mathcal{K})$, the authors in [32] proved and gave the conditions for which, the approximated kernel $\text{Viab}_{g^r}(\mathcal{K}_d)$ converges to the actual kernel $\text{Viab}_f(\mathcal{K})$ as d and ρ go to zero:

$$\lim_{d,\rho \to 0} \text{Viab}_{g^r}(\mathcal{K}_d) = \text{Viab}_f(\mathcal{K}) \tag{12}$$

The reader is referred to [32] for more details on the convergence issue, as well as the proof of the result stated in (9).

5 Conservative Viability Algorithm

Despite the convergence results mentioned in §4, the fact remains that the viability kernel $\text{Viab}_{g^r}(\mathcal{K}_d)$ is only an approximation of the exact viability kernel $\text{Viab}_f(\mathcal{K})$. The main issue is that this approximation is not conservative, *i.e.* certain states will be labeled by Algorithm 1 as belonging to $\text{Viab}_{g^r}(\mathcal{K}_d)$ when, in truth, they do not belong to $\text{Viab}_f(\mathcal{K})$. The non conservative nature of $\text{Viab}_{g^r}(\mathcal{K}_d)$ is due to the various discretization assumptions, both in time and space, that have been made in order to obtain the finite and discrete dynamical system (6).



(a) s_n and s_{n+1} are in $\mathcal{K}$ but the state trajectory in between is not.

(b) $s_{n+1} = g(s_n, u)$ is outside $\mathcal{K}$ but $g^r(s_n, u) \cap \mathcal{K}_d \neq \emptyset$.

(c) The state trajectory is viable by the algorithm standards but discontinuous in practice.

Figure 2: Approximation Issues of the Viability Algorithm.

To begin with, the time discretization of the continuous dynamical system (1) into (3) using an approximate method such as the Euler explicit scheme yields discrepancies between $\tilde{s}(s_0, u, \rho)$ and $g(s_0, u)$ for a starting state s_0. Such discrepancies directly affects the resulting viability kernel. To avoid this issue, one must resort to an exact time discretization of the system whenever possible.

Then, because of the time discretization, it may happen that both s_n and its successor s_{n+1} belong to $\mathcal{K}_d$, but the state trajectory in between leaves $\mathcal{K}$ (Fig. 2a). Addressing this issue is easy, it suffices to check whether the state trajectory between s_n and s_{n+1} satisfies the viability constraints.

Finally, recall from §4.1, the introduction of the hyperball $\mathcal{V}(r)$ to adapt (3) to the finite grid S_d. It yields two problems: the first one appears when a state s_n has a successor state $s_{n+1} = g(s_n, u)$ that does not belong to $\mathcal{K}$ but is such that $g^r(s_n, u) \cap \mathcal{K}_d \neq \emptyset$ (Fig. 2b). In this case, although s_n is non viable, it will not be discarded by Algorithm 1. The second problem stems from the fact that a state s_n does not have to reach another viable state s_{n+1} in order to classify as viable, it just has to get *close* to it. This results in discontinuous state trajectories that might be viable by the algorithm standards but that the actual system may not be able to follow in practice (Fig. 2c).

Algorithm 2: Conservative Viability Algorithm

Input: Exact discrete-time dynamical system g; State space lattice S_d; Discrete control space $\mathcal{U}_d$; Viability constraint set $\mathcal{K}$

Output: Conservative viability kernel $\text{Viab}_g(\mathcal{K}_d)$

$\mathcal{K}_d \leftarrow S_d \cap \mathcal{K}$;
$n \leftarrow 0$;
$\mathcal{K}^0 \leftarrow \mathcal{K}_d$;
repeat
  $\mathcal{K}^{n+1} \leftarrow \{s \in \mathcal{K}^n \mid \exists u \in \mathcal{U}_d(s) : g(s, u) \in \mathcal{K}^n \text{ and trajectory from } s \text{ to } g(s, u) \subset \mathcal{K}\}$
  $n \leftarrow n + 1$
until $\mathcal{K}^n = \mathcal{K}^{n+1}$;
return $\mathcal{K}^n$

From a viability point of view, it is critical to address these issues in order to obtain a conservative viability kernel $\text{Viab}_{g^r}(\mathcal{K}_d)$. To that end, it is first assumed that an exact time discretization of the system is available. Then, through an appropriate state space discretization, the need of the hyperball $\mathcal{V}(r)$ is relaxed: it is achieved by building a *state space lattice* based on the dynamical model of the system. A state space lattice is a prevalent structure in the field of robot motion planning [10, 29, 30] and it consists of a graph whose vertices represent a regular sampling of the state space and whose edges correspond to a carefully crafted set of controls (the case study presented in §7 details how the state space lattice is built for a double integrator system). Now, when S_d is a state space lattice, it is not necessary to extend g with $\mathcal{V}(r)$ since, by construction, $\forall s \in S_d, g(s, u) \in S_d$, and a conservative viability kernel $\text{Viab}_g(\mathcal{K}_d)$ can actually be computed using Algorithm 2, a slightly modified version of Algorithm 1.

6 Time-Varying Viability Constraints

The viability constraint set $\mathcal{K}$ has been defined as the compact subset of the state space S within which the dynamical system must remain. What happens now when the viability constraints are time-dependent? It is the case for instance when viability is related to the collision avoidance of obstacles that are moving. In robotics, one way to deal with moving obstacles is to cast the problem into the state-time space framework [11], *i.e.* to add time as an extra dimension to the state space. In this framework, the dynamical

system (3) can be rewritten:

$$\begin{cases} (s_{n+1}, \tau_{n+1}) = h((s_n, \tau_n), u_n) = (g(s_n, u_n), \tau_n + \rho) \\ u_n \in \mathcal{U}_d \end{cases} \tag{13}$$

where τ denotes time. In the state-time framework, it becomes possible to consider time-dependent viability constraints by defining $\mathcal{K}$ as the set of all the tuples (s, τ) that satisfy the viability constraints. Adapting Algorithm 2 so that it operates in state-time is straightforward.

One problem remains though, $\mathcal{K}$ has to be compact (recall that Algorithm 2 relies upon this assumption to converge). Upper-bounding the time dimension by setting a time horizon T_h allows to obtain a viability constraint set $\mathcal{K}$ which is compact. However, keeping in mind that a state is viable if it exists at least one sequence of controls that keeps the dynamical system in the viability constraint set *indefinitely*, it is obvious that, whatever the sequence of controls which is applied to the system from a given starting state-time, at some point in time, as soon as τ becomes greater than T_h, the state-time (s, τ) will leave $\mathcal{K}$ and the starting state-time will considered as nonviable. In this situation, Algorithm 2 would always return an empty set.

It is however possible to identify two classes of situations where the nature of the time-dependent viability constraints are such that it becomes possible to circumvent the problem stated above and to actually compute the viability kernel using Algorithm 2. The two classes of situations are respectively called *freezing* and *periodic*, they are presented in the next two sections.

6.1 Freezing Case

In this class of situation, it is assumed that the viability constraints stop being time-dependent at a given time T_f (imagine moving obstacles either leaving the environment or standing still after T_f). In other words, the following holds:

$$\forall \tau > T_f \ : \ (s, \tau) = (s, T_f) \tag{14}$$

In this case, the first step is to define the viability constraint set $\mathcal{K}$ as the set of all the tuples (s, τ) for which s satisfies the viability constraints and $\tau \leq T_f$, note that $\mathcal{K}$ is compact. The next step is to rewrite the dynamical system (13) as follows:

$$\begin{cases} (s_{n+1}, \tau_{n+1}) & = h((s_n, \tau_n), u_n) = (g(s_n, u_n), \tau_n + \rho) \text{ if } \tau_n < T_f \\ (s_{n+1}, \tau_{n+1}) & = h((s_n, \tau_n), u_n) = (g(s_n, u_n), \tau_n) \text{ if } \tau_n \geq T_f \\ u_n \in \mathcal{U}_d & \end{cases} \tag{15}$$

Under (15), it can be noted that, whatever the sequence of controls which is applied to the system from a given starting state-time, the time component of the state-time of the system will never be greater than T_f. It therefore becomes possible to compute the viability kernel of $\mathcal{K}$ using Algorithm 2.

6.2 Periodic Case

In this class of situation, it is assumed that the time-dependence of the viability constraints is periodic with a period T_p (imagine moving obstacles returning to their initial state and repeating the same motion over and over again). In other words, the following holds:

$$\forall \tau > T_p \ : \ (s, \tau) = (s, \tau \bmod T_p) \tag{16}$$

In this case, the first step is once again to define the viability constraint set $\mathcal{K}$ as the set of all the tuples (s, τ) for which s satisfies the viability constraints and $\tau \leq T_p$. The next step is to rewrite the dynamical

system (13) as follows:

$$\begin{cases}(s_{n+1},\tau_{n+1}) = h((s_n,\tau_n),u_n) = (g(s_n,u_n),(\tau_n+\rho) \bmod T_p)\\ u_n \in \mathcal{U}_d\end{cases} \tag{17}$$

Under (17), as in the freezing case, whatever the sequence of controls which is applied to the system from a given starting state-time, the time component of the state-time of the system will never be greater than T_p, and it is possible to compute the viability kernel of $\mathcal{K}$ using Algorithm 2.

7 Robotic Case Studies

From this point on, the paper investigates how viability and the viability algorithm can be used in robotic scenarios. The robotic system at hand and the corresponding state space lattice that the viability algorithm requires are respectively presented in §7.1 and §7.2. Various robotic viability constraints are then detailed in §7.3. Finally, Section §8 presents the results obtained on a number of scenarios featuring different sets of robotic viability constraints.

7.1 Robot Model

Let $\mathcal{A}$ denote a robotic system operating in a workspace $\mathcal{W}$. Henceforth, $\mathcal{A}$ denotes a 2D double integrator system whose acceleration a is directly controlled. A state s of $\mathcal{A}$ is represented by a tuple (p,v), where p is a 2D position, and v a Cartesian velocity. The motion of $\mathcal{A}$ is governed by:

$$\begin{cases}\dot{p} = v\\ \dot{v} = a\end{cases} \tag{18}$$

with $|v| \leq v_{max}$ and $|u| \leq a_{max}$. For a time step ρ, the following discrete state-transition equations are easily derived:

$$\begin{cases}p_{n+1} = p_n + v\rho + \frac{1}{2}a\rho^2\\ v_{n+1} = v_n + a\rho\end{cases} \tag{19}$$

Eq. (19) is the exact discrete-time version of the dynamical system (18), the equivalent of (3), that the conservative version of the viability algorithm requires. It is assumed that the body of the robot is a disk.

As simple as this robot model may appear, keep in mind that it is a second-order acceleration-controlled system. In this respect, it is more realistic than the first-order velocity-controlled systems that are sometimes used, *e.g.* the notorious "Dubins airplane" [9, 28].

7.2 State Space Lattice

For a fully-actuated system such as (19), a state space lattice can be built according to the method described in [10] which is outlined as follows. Let the set of possible controls $\mathcal{U}_d$ be restricted to $\{-a_{max}, 0, a_{max}\}$, and the time step ρ be chosen such that v_{max} is a multiple of $a_{max}\rho$. The term (u,ρ)-bang refers to applying a control $u \in \mathcal{U}_d$ for a duration ρ. Let $s_0 = (p_0, v_0)$ denote the *origin* state, the state space lattice S_d is the set of all states $s_i = (p_i, v_i)$ reachable from s_0 by a sequence of (u,ρ)-bangs. It is straightforward to establish that:

$$\begin{cases}p_i = p_0 + \frac{1}{2}m_i a_{max}\rho^2\\ v_i = v_0 + n_i a_{max}\rho\end{cases} \tag{20}$$

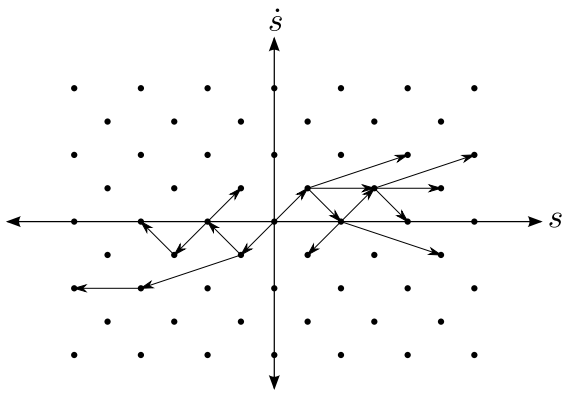


Figure 3: State space lattice for a 1D double integrator.

where $m_i, n_i \in \mathbb{N}$. Thus, S_d is a regular grid which has a spacing of $a_{max}\rho^2$ in position and $a_{max}\rho$ in velocity. Note that the grid positions p_i for odd multiples of $a_{max}\rho$ are offset by $\frac{1}{2}a_{max}\rho^2$ from the grid positions for even multiples of $a_{max}\rho$ (see Fig. 3 for a 1D system example). Note also that, in a space×time perspective, the lattice S_d has a constant spacing ρ in the time dimension.

This method can be applied to build a state-space lattice for arbitrary fully-actuated robotic systems. The case of under-actuated systems such as car-like vehicle is trickier to handle, however a number of solutions that could be used have been proposed, *e.g.* [24, 29, 30, 31, 38]. As soon as the state-lattice had been defined, Algorithm 2 can be applied.

7.3 Robotic Viability Constraints

To demonstrate the versatility of Viability, various kinds of robotic-related viability constraints are considered. The first kind is standard collision avoidance (§7.3.1). The second kind has to do with visibility, it becomes relevant as soon as the robotic system at hand is engaged in pursuit-evasion missions (§7.3.2). The third and last kind arises when the robotic system is subject to certain restrictions on its velocity, *e.g.* an airplane robot whose speed is lower-bounded (§7.3.3).

7.3.1 Collision Avoidance

Let us assume that $\mathcal{W}$ contains a set of b fixed and moving objects. Let $\mathcal{B}_i$ denote such an object, $\mathcal{B}_i(t)$ denotes the closed subset of $\mathcal{W}$ occupied by $\mathcal{B}_i$ at time t. Likewise, $\mathcal{B}_i([t_1, t_2])$ denotes the space×time region occupied by $\mathcal{B}_i$ during the time interval $[t1, t2]$. Note that $\mathcal{B}_i = \mathcal{B}_i([0, \infty))$. Let $\mathcal{B}$ denote the union of the workspace objects (both in space and time):

$$\mathcal{B} = \bigcup_{i=1}^{b} \mathcal{B}_i = \bigcup_{i=1}^{b} \mathcal{B}_i([0,\infty)) = \bigcup_{i=1}^{b} \bigcup_{t\in[0,\infty)} \mathcal{B}_i(t) \tag{21}$$

In viability terms, the viability constraint set within which $\mathcal{A}$ must be kept is the set of states where $\mathcal{A}$ is not in collision with any of the workspace obstacles:

$$\mathcal{K}_c = \{(s(t) \in S \mid \mathcal{A}(s(t)) \cap \mathcal{B}(t) = \emptyset\} \tag{22}$$

with $\mathcal{A}(s(t))$ the closed subset of the workspace $\mathcal{W}$ occupied by $\mathcal{A}$ when it is in the state $s(t)$.

7.3.2 Visibility

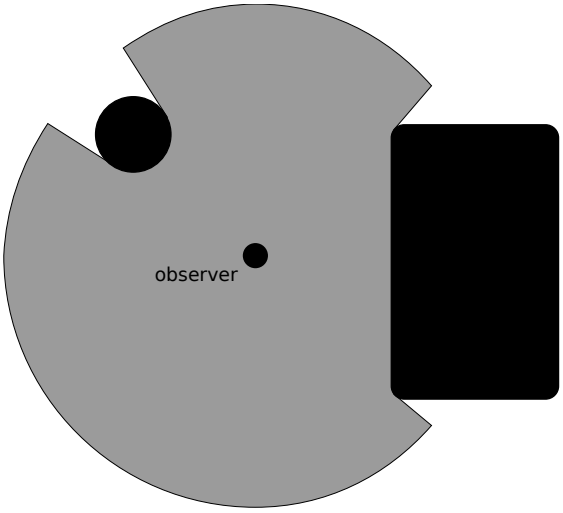


Figure 4: Field of view (grey area) for an observer among obstacles (black areas).

Visibility constraints do arise for robotic systems engaged in pursuit-evasion missions. Such missions generally feature at least an observer and a target. The observer is equipped with sensors allowing it to obtain information about a limited region of $\mathcal{W}$. The region of $\mathcal{W}$ that is known by the observer at a given state $s(t)$ is called its field of view and is denoted $\text{FoV}(s(t))$ and its shape depends on the type of sensors available. The unknown regions of $\mathcal{W}$ are either out of the sensors' range or occluded (Fig. 4).

The robotic system $\mathcal{A}$ can either endorse the role of the observer or the target. When $\mathcal{A}$ is the observer, it should always maintain one or more workspace targets $\mathcal{B}_i^t$ within its field of view at all times. The corresponding viability constraint set can be defined in this case as:

$$\mathcal{K}_v = \{s(t) \in S \mid \bigwedge_i \text{FoV}(s(t)) \cap \mathcal{B}_i^t(t) \neq \emptyset\} \tag{23}$$

Now, when $\mathcal{A}$ is the target, it should always stay out of the field of view of one or more workspace observers $\mathcal{B}_i^o$ at all times. The corresponding viability constraint set can be defined in this case as:

$$\mathcal{K}_v = \{s(t) \in S \mid \bigwedge_i \mathcal{A}(s(t)) \cap \text{FoV}(\mathcal{B}_i^o(t)) = \emptyset\} \tag{24}$$

Other variants could similarly be defined, *e.g.* the case where the robot target should always stay in the field of view of the workspace observers.

7.3.3 Velocity

Collision avoidance and visibility constraints impose restrictions on the configurations, *i.e.* positions and orientations, that the robotic system $\mathcal{A}$ can take. One could easily imagine situations where constraints are imposed on other components of the system's state, such as its velocity. An example of this would be if the system $\mathcal{A}$ is unstoppable, *e.g.* an airplane, or if $\mathcal{W}$ comprises regions with upper-bounded speed, *e.g.* the roadway. One could also imagine pursuit-evasion cases where an observer can only sense moving targets. In that case, standing still within the observer's field of view would be OK. All such constraints can readily be expressed via the definition of the corresponding viability constraint set.

In the end, no matter how different in nature the various constraints imposed on $\mathcal{A}$ are, they are expressed under the form of different viability constraint sets $\mathcal{K}_i$ and can all be merged to form the final viability constraint set $\mathcal{K}$ that will feed Algorithm 2:

$$\mathcal{K} = \bigcap_i \mathcal{K}_i \tag{25}$$

8 Simulation Results

To demonstrate its versatility and its ability to address different kinds of viability problems, the conservative viability algorithm 2 has been implemented for the case of the double integrator system and tailored to handle different workspaces, *i.e.* static/freezing/periodic, and different viability constraints, *i.e.* collision avoidance/visibility/velocity. It led to the definition of seven scenarios with mixed combinations of workspace and viability constraint types. The seven scenarios are described below where details about the workspace and the viability constraints are given. For each scenario, Algorithm 2 has been used to compute the conservative and discrete viability kernel $\text{Viab}_g(\mathcal{K}_d)$ and the corresponding regulation map R_d. The computed regulation map R_d then serves as a look-up table that indicates at each state the available controls that, when applied, will ensure that the system remains inside the viability kernel. A straightforward analysis of Algorithm 2 shows that its time complexity depends on the size of the discrete set of states and the discrete set of controls (line 5). In other words, it grows exponentially with the dimensions of the state and the control spaces. It is not really a problem since it should be kept in mind that the computation of $\text{Viab}_g(\mathcal{K}_d)$ and R_d is done *off-line* and *only once for each scenario*. In the current implementation (in Python on a average laptop), the running times for the different scenarios range from 6 to 20 minutes.

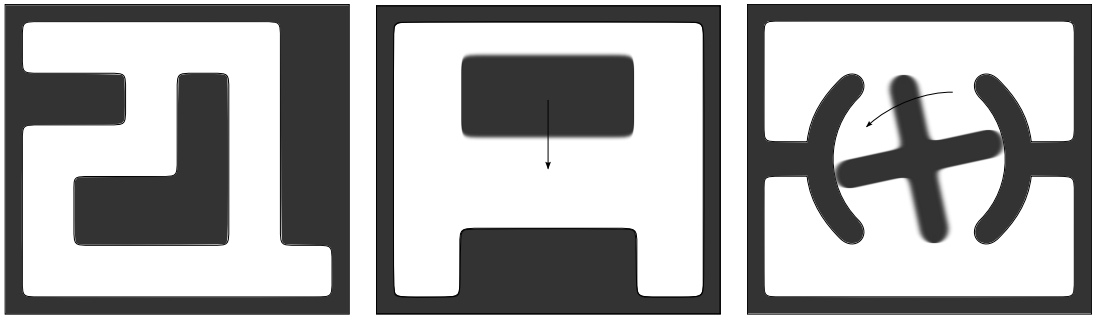
Algorithm 3: Safe Reactive Navigation

Input: Current state s_0; Discrete regulation map R_d ; Cost function $\mathcal{C}$
Output: Next control u^*
1 $u^* \leftarrow \text{argmin}_{u \in R_d(s_0)} \mathcal{C}(g(s_0, u))$;
2 **return** u^*

To demonstrate the usefulness of the computed $\text{Viab}_g(\mathcal{K}_d)$ and R_d, they have been used within an efficient on-line navigation scheme that is able to drive the system $\mathcal{A}$ around its workspace $\mathcal{W}$ while *always* respecting the different viability constraints at hand. The navigation scheme is rather simple and purely reactive: starting from an arbitrary state belonging to the viability kernel $\text{Viab}_g(\mathcal{K}_d)$, the navigation scheme selects, at each time step, the control to apply to $\mathcal{A}$ among the viable controls that are available at the current state. The set of viable controls is determined according to the regulation map R_d. The choice of the control depends on the task at hand and could be chosen randomly or so as to minimize a given cost function $\mathcal{C}$, *e.g.* distance to a goal. The control selection algorithm is outlined in Algorithm 3. It has been implemented using ROS[1] and GAZEBO[2]. For illustration purposes, three snapshots at different times of a typical simulation run[3] are given. Each snapshot depicts the workspace (black regions are obstacles), the system's current position and the trail of its trajectory. The 2D slice of the viability kernel corresponding to the current velocity is overlaid on the workspace: the viability kernel is shown in green and its complement in red. In the pursuit/evasion scenarios, the grey areas corresponds to the states where

[1] `http://www.ros.org`
[2] `http://gazebosim.org`
[3] Full videos available at `http://thierry.fraichard.free.fr/research`

the visibility constraints do not hold because of field of views. In all cases, the scenarios illustrate the ability of a purely reactive viability-based navigation scheme to control $\mathcal{A}$ forever without ever violating any of the viability constraints at hand.

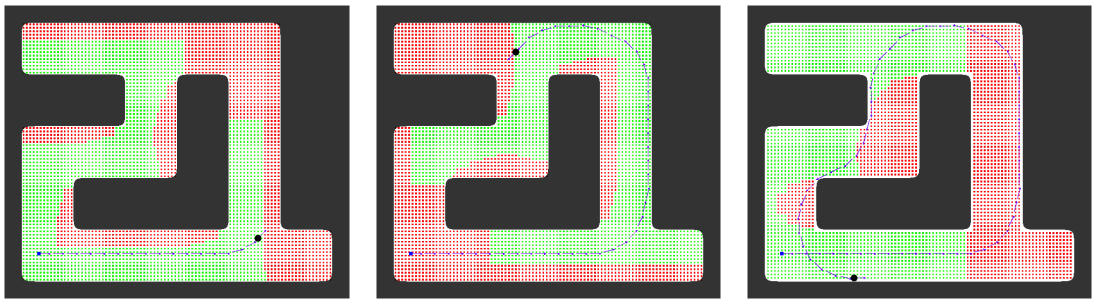


(a) Airplane scenario. (b) Compactor scenario. (c) Revolving door scenario.

Figure 5: Test workspace scenarios (from left to right: static, freezing and periodic cases).

8.1 Static Workspace

For this scenario, the workspace $\mathcal{W}$ contains static obstacles only (Fig. 5a). To emulate an airplane, the velocity of $\mathcal{A}$ is lower bounded: $v > v_{min}$. Besides, the upper bound on $\mathcal{A}$'s acceleration $|a| \leq a_{max}$ and the width of $\mathcal{W}$'s corridors are such that it prevents $\mathcal{A}$ from flying in circles at any given position in $\mathcal{W}$. In this scenario, the viability constraints are collision avoidance and velocity. The snapshots of Fig. 6 illustrate a typical simulation run.



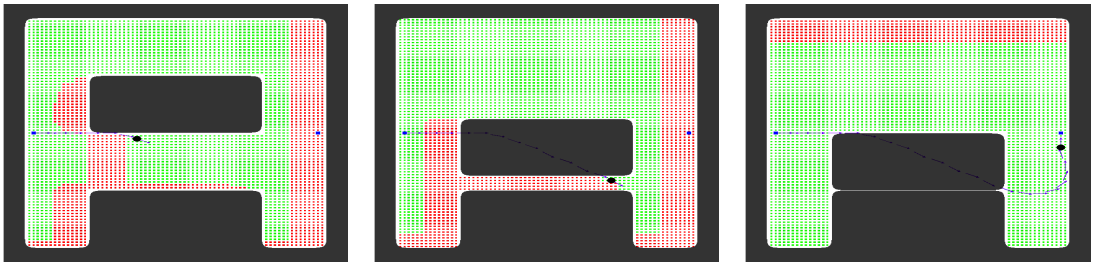
(a) $\tau = 16, v_x = 6, v_y = 6$. (b) $\tau = 39, v_x = -6, v_y = -6$. (c) $\tau = 59, v_x = 8, v_y = 0$.

Figure 6: 2D viability kernel slices of the airplane scenario at different times.

8.2 Freezing Workspace

For this scenario, the workspace $\mathcal{W}$ contains one static obstacle region and one moving obstacle that moves downward until it makes contact with the static obstacle, a behaviour resembling a trash compactor (Fig. 5b). The viability constraints are collision avoidance only. However, $\mathcal{A}$ has a goal now: it starts on the left side of the compactor and has to reach the right side. In this case, the choice of the control is not random anymore, the navigation scheme selects the control that will drive $\mathcal{A}$ closer to its goal. The

snapshots of Fig. 7 illustrate a typical simulation run. This scenario is not as simple as it appears, it is similar to the one discussed in [13]. It is a case where the ICS-based approaches have a hard time finding the right set of evasive trajectories.

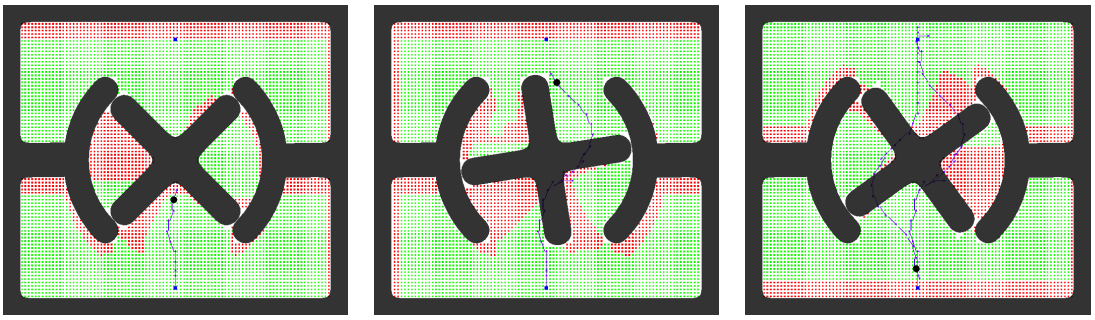


(a) $\tau = 8, v_x = 8, v_y = -4.$ (b) $\tau = 14, v_x = 8, v_y = -6.$ (c) $\tau = 23, v_x = 0, v_y = 8.$

Figure 7: 2D viability kernel slices of the compactor scenario at different times.

8.3 Periodic Workspace

8.3.1 Collision Avoidance



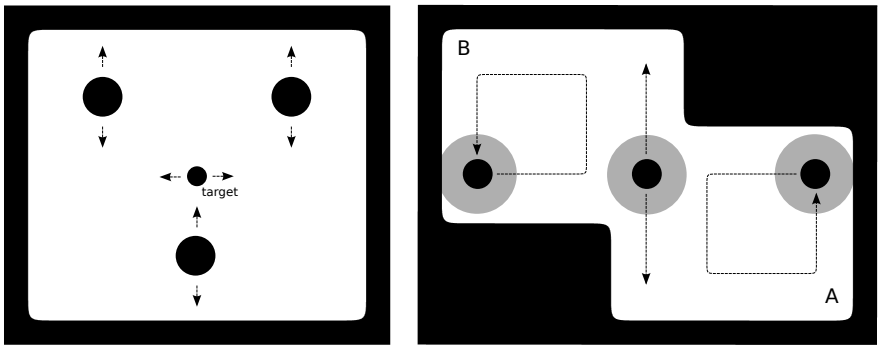
(a) $\tau = 10, v_x = 2, v_y = 6.$ (b) $\tau = 29, v_x = -4, v_y = 6.$ (c) $\tau = 59, v_x = 2, v_y = -6.$

Figure 8: 2D viability kernel slices of the revolving door scenario at different times.

For this scenario, the workspace $\mathcal{W}$ contains both static and moving obstacles, it comprises two "rooms" and the only way to pass from one to the other is to use a revolving door (Fig. 5c). The revolving door has constant angular velocity and its behavior is periodic. The viability constraints are collision avoidance only. Now, $\mathcal{A}$ has a task to accomplish which is to repeatedly pass from one room to the other. To that end, two goal positions are respectively defined in both rooms: when the current goal is reached, the other goal becomes the current goal and so forth. The snapshots of Fig. 8 illustrate a typical simulation run.

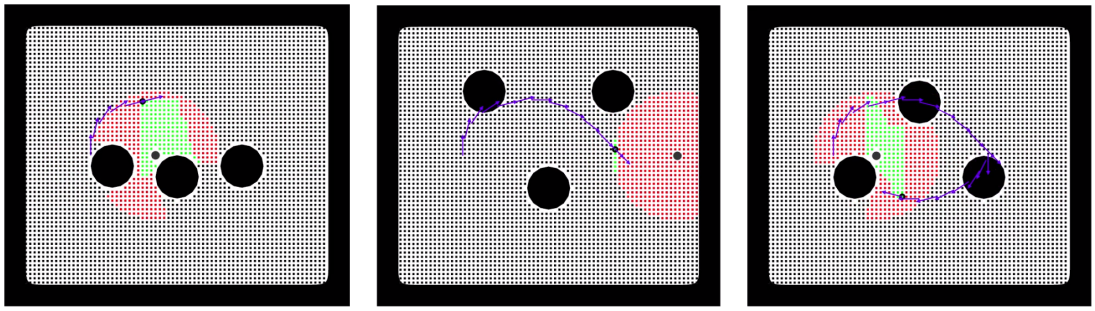
8.3.2 Pursuit

For this scenario, the workspace $\mathcal{W}$ contains both static and moving obstacles. All moving obstacles are assumed to have a periodic behavior (Fig. 9a). The system $\mathcal{A}$ is equipped with an omni-directional sensor and its task now is to keep one of the moving obstacles, the *target*, within its field of view at all times



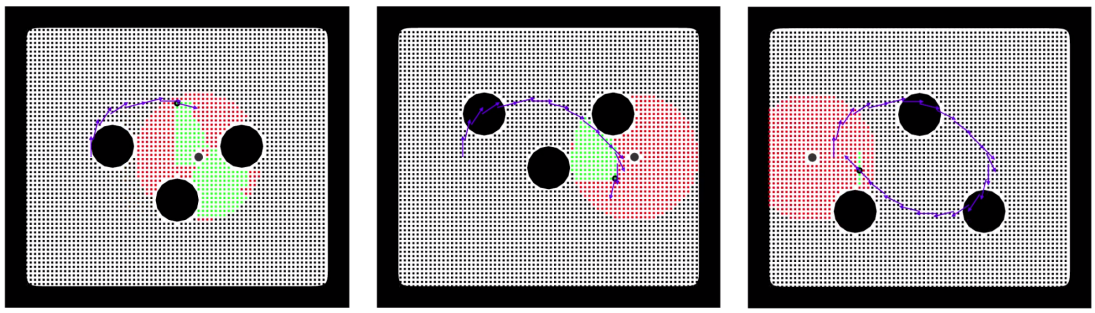
(a) Pursuit scenario. (b) Evasion scenario.

Figure 9: Periodic workspace scenarios with visibility constraints.



(a) $\tau = 5, v_x = 8, v_y = 2.$ (b) $\tau = 12, v_x = 2, v_y = 2.$ (c) $\tau = 20, v_x = 8, v_y = 2.$

Figure 10: 2D viability kernel slices of the pursuit scenario at different times.

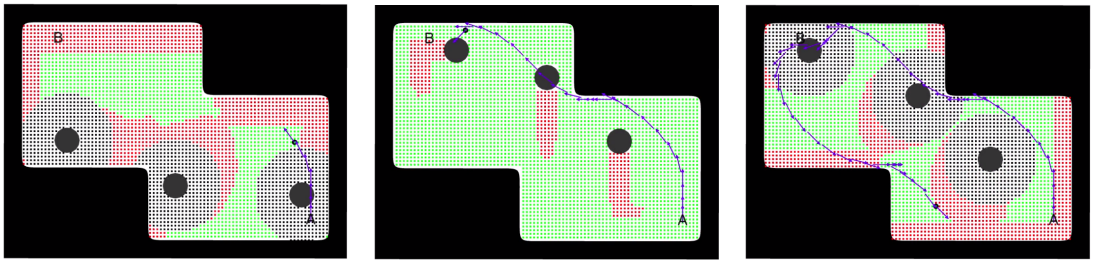


(a) $\tau = 7, v_x = 8, v_y = -2.$ (b) $\tau = 14, v_x = -2, v_y = -8.$ (c) $\tau = 23, v_x = -4, v_y = 4.$

Figure 11: 2D viability kernel slices of the unstoppable pursuit scenario at different times.

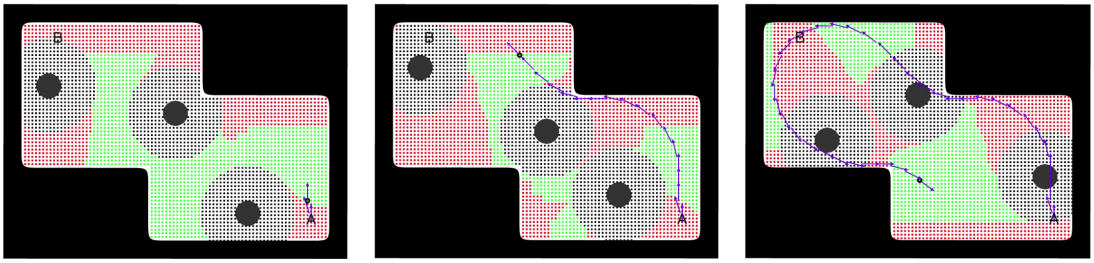
while avoiding collisions of course. The viability constraints are collision avoidance and visibility. The snapshots of Fig. 10 illustrate a typical simulation run. In this case, the navigation scheme was set to select the control maximizing the distance to the target. For an extra challenge, an additional constraint in the form of a lower-bound on the system's velocity is considered, and the corresponding results are depicted in Fig. 11.

8.3.3 Evasion



(a) $\tau = 6, v_x = -6, v_y = 8.$ (b) $\tau = 30, v_x = 0, v_y = 0.$ (c) $\tau = 63, v_x = 6, v_y = -6.$

Figure 12: 2D viability kernel slices of the evasion scenario at different times.



(a) $\tau = 2, v_x = 0, v_y = 8.$ (b) $\tau = 17, v_x = -8, v_y = 8.$ (c) $\tau = 39, v_x = 8, v_y = -6.$

Figure 13: 2D viability kernel slices of the unstoppable evasion scenario at different times.

For this scenario, the workspace $\mathcal{W}$ contains both static and moving obstacles. The moving obstacles have a periodic behavior, they are assumed to be *sentinels* on patrol duty, they are equipped with omni-directional sensors with a limited field of view (Fig. 9b). To make things more interesting, it is further assumed that the sensors can only detect moving objects. The system $\mathcal{A}$ is tasked to navigate from point A to point B and back without colliding with the workspace obstacles or being detected by the sentinels. The viability constraints are collision avoidance, visibility, and velocity. The regulation map corresponding to this scenario allowed $\mathcal{A}$ to complete the task with success, even with a navigation policy as simple as choosing at each step the control that minimizes the distance to the goal. The snapshots of Fig. 12 illustrate a typical simulation run. As with the pursuit scenario, the case where the velocity of $\mathcal{A}$ is lower-bounded has also been considered, the corresponding results are depicted in Fig. 13.

9 Discussion and Conclusion

Guaranteeing safe, *i.e.* collision-free, motion for robotic systems is usually tackled in the Inevitable Collision State (ICS) framework. This paper has explored the use of the Viability framework to address the more general problem of guaranteeing safe robot motion when it involves more than mere collision avoidance. It has first proposed the Conservative Viability Algorithm which is able to compute off-line the viability kernel and the regulation map for a robotic system. In a second stage, it has demonstrated, for seven very different scenarios, how to use the computed viability kernel and the regulation map within an on-line reactive navigation scheme that can drive the robotic system without ever violating the motion constraints at hand (collision avoidance, visibility, velocity).

The algorithm proposed is conservative and can handle time-varying motion constraints such as moving obstacles. Although it is in general impossible to compute a non-empty viability kernel in the presence of moving obstacles, two classes of dynamic environments have been identified, freezing and periodic, for which it is possible to compute a valid viability kernel. Accordingly, it becomes possible to guarantee motion safety in the presence of moving obstacles for these two classes of environments. Although guaranteed collision avoidance has already been demonstrated for freezing environments using Inevitable Collision States, it is the first time that a similar result is achieved for periodic environments.

The Viability framework is definitely an interesting alternative to the ICS framework because of its ability to handle different motion constraints in a unified manner. However, this feature comes at the cost of a greater computational complexity that prevents the on-line computation of the viability kernel and the regulation map. Depending on the task at hand, this may or may not be an issue. Note however that a more efficient implementation of the algorithm proposed could be obtained through parallel computing [7].

Besides the efficiency issue, future works include exploring how the Conservative Viability Algorithm whose formulation is general can handle alternative robotic systems. At a more fundamental level, further investigation into non-freezing and non-periodic environments should be carried out in order to determine whether viability can nonetheless be useful when it comes to guaranteed motion safety in arbitrary dynamic environments.

References

[1] Aubin, J.P., Bayen, A., Saint-Pierre, P.: Viability Theory: New Directions. Springer (2011)

[2] Bekris, K., Kavraki, L.: Greedy but safe replanning under kinodynamic constraints. In: IEEE Int. Conf. on Robotics and Automation (ICRA). Roma (IT) (2007). DOI 10.1109/ROBOT.2007.363069

[3] Blaich, M., Weber, S., Reuter, J., Hahn, A.: Motion safety for vessels: An approach based on inevitable collision states. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). Hamburg (DE) (2015). DOI 10.1109/IROS.2015.7353504

[4] Bohórquez, N., Sherikov, A., Dimitrov, D., Wieber, P.B.: Safe navigation strategies for a biped robot walking in a crowd. In: IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids). Cancun (MX) (2016). DOI 10.1109/HUMANOIDS.2016.7803304

[5] Bouguerra, M., Fraichard, T., Fezari, M.: Safe motion using viability kernel. In: IEEE Int. Conf. Robotics and Automation (ICRA). Seattle (US) (2015). URL `http://hal.inria.fr/hal-01143861`

[6] Bouraine, S., Fraichard, T., Salhi, H.: Provably safe navigation for mobile robots with limited field-of-views in dynamic environments. Autonomous Robots **32**(3) (2012). DOI 10.1007/s10514-011-9258-8

[7] Brias, A., Mathias, J.D., Deffuant, G.: Accelerating viability kernel computation with cuda architecture: application to bycatch fishery management. Computational Management Science **13**(3), 371–391 (2016). DOI 10.1007/s10287-015-0246-x. URL https://doi.org/10.1007/s10287-015-0246-x

[8] Chan, N., Kuffner, J., Zucker, M.: Improved motion planning speed and safety using regions of inevitable collision. In: CISM-IFToMM symposium on robot design, dynamics, and control (2008)

[9] Chitsaz, H., LaValle, S.M.: Time-optimal paths for a Dubins airplane. In: Proceedings IEEE Conference Decision and Control. New Orleans, LA (US) (2007)

[10] Donald, B., Xavier, P., Canny, J., Reif, J.: Kinodynamic motion planning. Journal of the ACM (JACM) **40**(5) (1993). DOI 10.1145/174147.174150

[11] Fraichard, T.: Trajectory planning in a dynamic workspace: a state-time space approach. Advanced Robotics **13**(1) (1998). DOI 10.1163/156855399X00928. URL http://hal.inria.fr/inria-00259321

[12] Fraichard, T., Asama, H.: Inevitable collision states. a step towards safer robots? Advanced Robotics **18**(10) (2004). DOI 10.1163/1568553042674662

[13] Fraichard, T., Howard, T.: Iterative motion planning and safety issue. In: A. Eskandarian (ed.) Handbook of Intelligent Vehicles. Springer (2012)

[14] Frazzoli, E., Dahleh, M., Feron, E.: Real-time motion planning for agile autonomous vehicles. Journal of Guidance, Control, and Dynamics **25**(1) (2002). DOI 10.2514/2.4856

[15] Hsu, D., Kindel, R., Latombe, J.C., Rock, S.: Randomized kinodynamic motion planning with moving obstacles. Int. Journal of Robotics Research (IJRR) **21**(3) (2002). DOI 10.1177/027836402320556421

[16] Kalisiak, M., van de Panne, M.: Approximate safety enforcement using computed viability envelopes. In: IEEE Int. Conf. on Robotics and Automation (ICRA). New Orleans (US) (2004). DOI 10.1109/ROBOT.2004.1302392

[17] Kalisiak, M., van de Panne, M.: Faster motion planning using learned local viability models. In: IEEE Int. Conf. on Robotics and Automation (ICRA). Roma (IT) (2007). DOI 10.1109/ROBOT.2007.363873

[18] Korda, M., Henrion, D., Jones, C.: Convex computation of the maximum controlled invariant set for polynomial control systems. SIAM Journal on Control and Optimization **52**(5) (2014). DOI 10.1137/130914565

[19] Liniger, A., Lygeros, J.: Real-time control for autonomous racing based on viability theory. arXiv preprint arXiv:1701.08735 (2017). URL https://arxiv.org/abs/1701.08735

[20] Lygeros, J.: On reachability and minimum cost optimal control. Automatica **40**(6) (2004). DOI 10.1016/j.automatica.2004.01.012

[21] Macek, K., Vasquez, D., Fraichard, T., Siegwart, R.: Towards safe vehicle navigation in dynamic urban scenarios. Automatika **50**(3-4) (2009). URL http://hal.inria.fr/inria-00447452

[22] Maidens, J., Kaynama, S., Mitchell, I., Oishi, M., Dumont, G.: Lagrangian methods for approximating the viability kernel in high-dimensional systems. Automatica **49**(7) (2013). DOI 10.1016/j.automatica.2013.03.020

[23] Martinez-Gomez, L., Fraichard, T.: An efficient and generic 2d inevitable collision state-checker. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). Nice (FR) (2008). DOI 10.1109/IROS.2008.4650640. URL `http://hal.inria.fr/inria-00293508`

[24] McNaughton, M., Urmson, C., Dolan, J., Lee, J.W.: Motion planning for autonomous driving with a conformal spatiotemporal lattice. In: IEEE Int. Conf. on Robotics and Automation (ICRA) (2011)

[25] Mitchell, I., Bayen, A., Tomlin, C.: A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. IEEE Trans. on Automatic Control **50**(7) (2005). DOI 10.1109/TAC.2005.851439

[26] Mitsch, S., Ghorbal, K., Platzer, A.: On provably safe obstacle avoidance for autonomous robotic ground vehicles. In: Robotics: Science and Systems (RSS) (2013). URL `http://repository.cmu.edu/compsci/2694/`

[27] Monnet, D., Ninin, J., Jaulin, L.: Computing an inner and an outer approximation of the viability kernel. Reliable Computing **22** (2016). URL `https://hal.archives-ouvertes.fr/hal-01366752`

[28] Owen, M., Beard, R., McLain, T.: Implementing dubins airplane paths on fixed-wing uavs. In: Handbook of Unmanned Aerial Vehicles. Springer (2014)

[29] Pancanti, S., Pallottino, L., Salvadorini, D., Bicchi, A.: Motion planning through symbols and lattices. In: IEEE Int. Conf. on Robotics and Automation (ICRA). New Orleans (US) (2004)

[30] Pivtoraiko, M., Knepper, R., Kelly, A.: Differentially constrained mobile robot motion planning in state lattices. Journal of Field Robotics **26**(3) (2009). DOI 10.1002/rob.20285

[31] Rufli, M., Siegwart, R.: On the design of deformable input-/state- lattice graphs. In: IEEE Int. Conf. on Robotics and Automation (ICRA) (2010)

[32] Saint-Pierre, P.: Approximation of the viability kernel. Applied Mathematics and Optimization **29**(2) (1994). DOI 10.1007/BF01204182

[33] Savino, G., Giovannini, F., Fitzharris, M., Pierini, M.: Inevitable collision states for motorcycle-to-car collision scenarios. IEEE Trans. on Intelligent Transportation Systems **17**(9) (2016). DOI 10.1109/TITS.2016.2520084

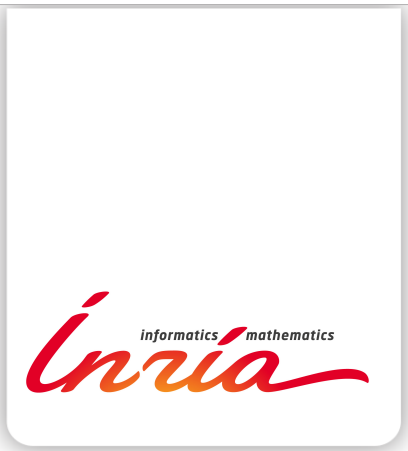
[34] Schouwenaars, T., How, J., Feron, E.: Receding horizon path planning with implicit safety guarantees. In: American Control Conference. Boston (US) (2004)

[35] Seder, M., Petrovic, I.: Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In: IEEE Int. Conf. on Robotics and Automation (ICRA) (2007)

[36] She, Z., Xue, B.: Computing an invariance kernel with target by computing lyapunov-like functions. IET Control Theory and Applications **7**(15) (2013). DOI 10.1049/iet-cta.2013.0275

[37] Shiller, Z., Gal, O., Raz, A.: Adaptive time horizon for on-line avoidance in dynamic environments. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). San Francisco (US) (2011). DOI 10.1109/IROS.2011.6094643

[38] Ziegler, J., Stiller, C.: Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios. In: IEEE/RSJ International Conference on Intelligent Robots and Sys- tems. (2009)



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399